AMENDMENTS TO THE CLAIMS

Please amend the claims to be as follows, where markings are included to show changes made.

1. (currently amended) A method of generating a software program executable binary file, the method comprising:
accessing a first file <u>for a first module</u> including source code therein;
accessing a second file <u>for a second module</u> including object code therein and further including object file summary information; and
generating the executable binary file from at least the first and second files,
<u>wherein the object file summary information includes a summary intermediate representation (SIR) and an extension to a linker symbol table, and</u>
wherein the object file summary information is used in optimizing the executable binary file generated.

2. (original) The method of claim 1, further comprising disambiguating memory accesses otherwise considered aliased using the object file summary information.

3. (canceled)

4. (currently amended) The method of ~~claim 3~~ <u>claim 1</u>, wherein the extension to the linker symbol table includes a flag indicating whether a procedure exposes a memory address by storing the address in a location accessible outside the procedure.

5. (currently amended) The method of ~~claim 3~~ claim 1, wherein the SIR includes a summary symbol table.

6. (currently amended) The method of claim 5, wherein the summary symbol table includes global and static symbols accessed in [[the]] a procedure, formal parameters of the procedure, return location for the procedure, and other procedures called by the procedure.

7. (currently amended) The method of claim 6, wherein a symbol is referenced in the summary symbol table [[in]] by using an associated summary symbol identifier (SYMID).

8. (original) The method of claim 7, wherein a symbol entry includes a linker identifier (LI_ID) of the entry from a linker symbol table.

9. (original) The method of claim 5, wherein the SIR uses an operator for memory referencing.

10. (currently amended) The method of claim 5, wherein the SIR uses an operator to adjust [[the]] an address expression by an offset.

11. (original) The method of claim 5, wherein the SIR uses an operator to take an address of a function or variable.

12. (original) The method of claim 5, wherein the SIR uses an operator to merge pointer values from different control flow paths.

13. (original) The method of claim 5, wherein the SIR uses an operator to represent direct procedure calls.

14. (original) The method of claim 5, wherein the SIR uses an operator to represent indirect procedure calls.

15. (original) The method of claim 5, wherein the SIR uses a no-operation type operator to discard values.

16. (original) The method of claim 5, wherein the SIR includes a control data structure comprising a link field for each procedure that points to an SIR block of a next procedure.

17. (original) The method of claim 5, wherein the SIR includes a control data structure comprising a table having links to an SIR block for each procedure.

18. (original) The method of claim 1, further comprising determining variables modified by and referenced by function calls in the object code using the object file summary information.

19. (cancelled)

20. (currently amended) The method of ~~claim 19~~ claim 18, wherein the extension to the linker symbol table includes a first flag indicative of whether a procedure modifies non-local variables and a second flag indicative of whether the procedure references non-local variables.

21. (original) The method of claim 20, wherein the extension to the linker symbol table includes a second flag indicative of whether the procedure modifies global/static variables excluding callees and a third flag indicative of whether the procedure references non-local variables excluding callees.

22. (currently amended) The method of ~~claim 19~~ claim 18, wherein the per-procedure summary data comprises a linked list of entries corresponding to symbols directly ~~mod-ref'd~~ modified or referenced in a procedure.

23. (original) The method of claim 22, wherein each entry comprises a linker identifier of a corresponding symbol and flags indicative of whether that symbol is modified or referenced.

24. (original) The method of claim 1, wherein the second file comprises a load module that is a shared library of procedures.

25. (original) The method of claim 1, wherein multiple files including object code are accessed and used in compiling the program.

26. (currently amended) A system for generating a software program executable file, the system comprising:

a processing device configured to execute computer-readable program code;

a memory system configured to store the computer-readable program code and data;

a source file for a first module comprising source code stored by the memory system ~~for the program~~;

an object file for a second module including computer-readable program code and object file summary information; and

a translator comprising computer-readable program code stored by the memory system, wherein the computer-readable program code of the translator is configured to access at least the source and object files and to generate the executable file of the program therefrom,

wherein the object file summary information includes a summary intermediate representation (SIR) and an extension to a linker symbol table, and

wherein the object file summary information is used in optimizing the executable file generated.

27. (original) The system of claim 26, further comprising a points-to analyzer that uses the object file summary information to disambiguate memory accesses otherwise considered aliased.

28. (original) The system of claim 26, further comprising a module that uses the object file summary information to determine variables modified by and referenced by function calls in the object file.

29. (original) The system of claim 26, wherein the translator comprises:
a compiler configured to translate source files into intermediate files; and
a linker configured to access the object file summary information and communicate information to the compiler relevant to optimizing compilation of the program.

30. (original) The system of claim 29, wherein the translator further comprises a feedback provider that provides a communications interface between the compiler and the linker.

31. (currently amended) A computer-readable storage medium storing an [[An]] object file of a computer programming module, the ~~object file~~ computer-readable storage medium comprising:
computer-readable object code for the module; and
computer-readable object file summary information including a summary intermediate representation (SIR) and an extension to a linker symbol table for use by a compiler in optimizing executable code including the module.

32. (currently amended) The ~~object file~~ computer-readable storage medium of claim 31, wherein the SIR includes a summary symbol table.

33. (currently amended) The ~~object file~~ computer-readable storage medium of claim 32, wherein the summary symbol table includes global and static symbols accessed in the module, formal parameters of the module, return location for the module, and other procedures called by the module.

34. (currently amended) The ~~object file~~ computer-readable storage medium of claim 31, wherein the SIR uses a plurality of operators from a group of operators including an operator for memory referencing, an operator to adjust the address expression by an offset, an operator to take an address of a function or variable, an operator to merge pointer values from different control flow paths, an operator to represent direct procedure calls, an operator to represent indirect procedure calls, and a no-operation type operator to discard values.

35. (currently amended) The ~~object file~~ computer-readable storage medium of claim 31, wherein the SIR includes a linked list of entries corresponding to symbols directly ~~modified/referenced~~ modified or referenced in a procedure.